Rothermel, G., and Harrold, M. J. 1996. Analyzing regression test selection techniques. *IEEE Transactions on Software Engineering* 22(8): 529–551.

Sherlund, B., and Korel, B. 1995. Logical modification oriented software testing. *Proceedings: Twelfth International Conference on Testing Computer Software* June.

Taha, A. B., Thebaut, S. M., and Liu, S. S. 1989. An approach to software fault localization and revalidation based on incremental data flow analysis. *Proceedings of the 13th Annual International Computer Software and Applications Conference* September, 527–534.

White, L. J., and Leung, H. K. N. 1992. A firewall concept for both control-flow and data-flow in regression integration testing. *Proceedings of the Conference on Software Maintenance—1992* November, 262–270.

White, L. J., Narayanswamy, V., Friedman, T., Kirschenbaum, M., Piwowarski, P., and Oha, M. 1993. Test manager: a regression testing tool. *Proceedings of the Conference on Software Maintenance—1993* September, 338–347.

Yau, S. S., and Kishimoto, Z. 1987. A method for revalidating modified programs in the maintenance phase. *COMPSAC '87: The Eleventh Annual International Computer Software and Applications Conference* October, 272–277.

Ziegler, J., Grasso, J. M., and Burgermeister, L. G. 1989. An Ada based real-time closed-loop integration and regression test tool. *Proceedings of the Conference on Software Maintenance* October, 81–90.

Lessons Learned from a Regression Testing Case Study

DAVID S. ROSENBLUM dsr@ics.uci.edu
Department of Information and Computer Science, University of California Irvine, Irvine, CA 92697-3425

ELAINE J. WEYUKER weyuker@research.att.com
AT&T Labs-Research, 180 Park Avenue, Florham Park, NJ 07932

1. Introduction

The designer of an empirical study of a software maintenance algorithm, tool, method or process is faced with a number of unique problems. As with any empirical study of software, the study designer must strive to select representative software systems and representative development personnel as subjects. However, the designer of a maintenance study also faces the problem that the study typically must be carried out on multiple versions of the subject system or systems. Not only must a binary copy of each version be available for study, but it is also typically necessary to have access to the source code and test suites for each version (to allow analysis and/or instrumentation).

Regression testing is an aspect of software maintenance that has been studied intensively over the last two decades. Much of this research has involved the design of strategies for *selective regression testing*, which attempt to reduce the cost of regression testing by eliminating unnecessary test cases whenever changes to a system must be tested. Despite the fact that numerous selective regression testing strategies have been proposed over the last two decades, to date there has been little evidence presented to indicate that they are truly cost-effective. The first author recently carried out a case study to evaluate the cost-effectiveness of an automated code-based selective regression testing strategy called TESTTUBE (Chen, Rosenblum and Vo, 1994). The case study was carried out on a sequence of 31 versions of the 1988 release of the KornShell (KSH88), a popular command processor for the UNIX® operating system (Bolsky and Korn, 1995).[1] The data obtained in this study led to our recent investigation of methods for predicting the cost-effectiveness of regression testing strategies (Rosenblum and Weyuker, 1996). This paper describes the design of the case study and discusses the lessons that were learned in carrying it out.

2. Case Study Design

KSH88 exists in 59 versions, dating from May 27, 1988, to February 24, 1993 (after which the KornShell was rewritten and developed under the name KSH93). These versions were originally made available in a public tool database from which the latest version could be obtained for distribution to other organizations within AT&T. For the purposes of the case study, these 59 versions were all viewed as being official releases that required regression testing prior to distribution. Of these 59 versions, 31 were archived with their test suites, and it is these latter 31 versions that were studied. The test suite for each version is a collection of 13 to 16 shell scripts (programs of command input to be processed by the shell), each of which contains an average of 64 lines of shell input and tests a major subsystem of the shell (built-in commands, aliases, variable expansion, etc.).

The case study was carried out by applying the TESTTUBE tools during a simulation of the development history of the 31 versions. Each version was instrumented, built, regression tested with the test units selected by TESTTUBE, and then subjected to coverage analysis for the next version. During this simulated development, a large amount of data was gathered. The data were then analyzed to study how well TESTTUBE reduced the number of test cases needed to test each version, its cost-effectiveness in doing so, and the extent to which these results were impacted by the unique characteristics of the KornShell and its test suite.

One key result of the study was that in 80% of the versions, 100% of the test suite was selected, meaning that the analysis carried out by TESTTUBE was entirely wasted, regardless of how efficient that analysis was. Another key result was that the cost of the analysis carried out by TESTTUBE was two orders of magnitude higher than the break-even point for the average selection rate that was observed over the 31 versions. It should be noted that TESTTUBE is one of the most efficient of the selective regression testing methods that have been proposed, in terms of the cost of the analysis it performs. However, this efficiency is achieved by sacrificing some precision in the analysis.

3. Lessons Learned

KSH88 had a number of attributes that made it an attractive candidate for study:

- Both the source code *and* the test suite for each version were available for study.

- A large number of versions were available, allowing a more realistic assessment of long-term, average-case behavior.
- It is a real system of modestly large size with a large user community, not a small, contrived program example.
- The changes made to the 31 versions were real changes that were made to remove real faults or add new features, not hypothetical changes made to remove seeded faults.
- The creator of KSH88, Dave Korn, was a member of our organization and thus was available for consultation in case questions or problems arose during the study.

The first three of these attributes contributed significantly to making the case study realistic and its results meaningful for use by others. However, there were also a number of drawbacks to using KSH88 as a subject for study. Some of these drawbacks are simply characteristics of KSH88 that limit its utility as a representative example of software systems:

- KSH88 was developed by a single, unusually-talented programmer.
- Because the test suites for KSH88 were designed to be fully automated, there are major subsystems of KSH88 that are not tested by the test suites (especially the interactive features, such as command-line editing), making the coverage of the test suites rather low and non-uniform among features.
- KSH88 has the property that even very small inputs cause all of the components of the system to be exercised (scanner, parser, symbol table, semantic analyzer, back end), making it potentially ill-suited to the use of a selective regression testing method.

Still other drawbacks represent problems that must be faced by any designer of a maintenance study:

- Notwithstanding the traditional view of maintenance as comprising all development activities following the first release of a system, the distinction between maintenance and initial development is not always clear-cut. This is an important consideration in studies of selective regression testing methods because these methods have been proposed specifically as a way of reducing maintenance costs rather than as a way of reducing development costs in general (since initial development on a system may well involve changes to all parts of its source code). For instance, some of the later versions of KSH88 involved feature changes of such scope and size as to have the character of initial development.
- The KSH88 test suite contained a relatively small number of relatively large test cases, rather than a large number of small test cases. The granularity of the test suite can have a significant impact on the results of a maintenance study, so in general it may be desirable to break large test cases into smaller ones (although there was no obvious way of breaking the shell scripts in the KSH88 test suite into smaller units).

Despite these drawbacks, KSH88 provided an opportunity to discover weaknesses and opportunities for improvement in TESTTUBE. Additionally, it led to the identification of factors affecting the cost-effectiveness of selective regression testing methods in general, resulting in our subsequent research on the cost-effectiveness of regression testing strategies and the use of coverage-based predictors to determine the likely usefulness of a regression testing strategy.

Finally, KSH88 provided a much more realistic picture of the strengths, limitations and applicability of selective regression testing as a means of reducing maintenance costs. In contrast, most previously published empirical data on selective regression testing strategies involved hypothetical changes to single versions of small programs (e.g., see Harrold et al. (1993) or Bates and Horwitz (1993)). The only other extensive case studies of which we are aware are the Rothermel and Harrold studies (Rothermel and Harrold, 1996), which involved evaluations of a method called DejaVu on a number of software systems, including a 50,000-line computer game. For some of the systems they studied DejaVu was indeed a cost-effective method of selecting test cases, while for other systems it was not cost-effective.

Thus, the results for selective regression testing are mixed—cost-effectiveness is by no means guaranteed, and it depends heavily on the choice and design of the test selection method, the nature of the system being tested, the test suite design, the nature of the test coverage relation, and other factors. More empirical studies are needed, including comparative studies of different methods applied to the same system under test, in order to further reveal the ways in which these different factors influence the cost-effectiveness of a selective regression testing method.

Notes

1. UNIX is a registered trademark licensed exclusively by Novell, Inc.

References

Bates, S., and Horwitz, S. 1993. Incremental program testing using program dependence graphs. *Proc. 20th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* Charleston, SC, 384–396.

Bolsky, M. I., and Korn, D. G. 1995. *The New KornShell Command and Programming Language*. Prentice-Hall.

Chen, Y.-F., Rosenblum, D. S., and Vo, K.-P. 1994. TestTube: A system for selective regression testing. *Proc. 16th Int'l. Conf. on Software Engineering* Sorrento, Italy, 211–220.

Harrold, M. J., Gupta, R., and Soffa, M. L. 1993. A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology*, 2(3): 270–285.

Rothermel, G., and Harrold, M. J. 1994. A framework for evaluating regression test selection techniques. *Proc. 16th Int'l. Conf. on Software Engineering* Sorrento, Italy, 201–210.

Rothermel, G., and Harrold, M. J. 1996. A safe, efficient regression test selection technique. Technical Report OSU-CISRC-4/96-TR25, Department of Computer and Information Science, The Ohio State University.

Rosenblum, D. S., and Weyuker, E. J. 1996. Predicting the cost-effectiveness of regression testing strategies. *Proc. ACM SIGSOFT '96 Fourth Symposium on the Foundations of Software Engineering* San Francisco, CA, 118–126.